Nullability

In Kotlin we differentiate between references that can be null and those that can‘t be. This concept leads to some changes in how you write your code.

```
var a = 1
a = null
```

// Compile time error

```
var a : Int?  = 1
a = null
```

// No compile error

*Be aware that **Int** is a **class** whereas **Int?** is a **type**.*

We have different options in how we can deal with nullable values, which are shown on the following slides.

Nullability

Variant 1: Check explicitly for null (The java way)

```
val a : Int? = 2

val c = if(a!= null) a else -1
```

```
val d = a ?: -1
```

Variant 2: Safe Call operator

```
val a : String? = "Hello"
val b = a.length
```

// Compile time error

```
val a : String? = "Hello"
val b = a?.length
```

Nullability

Variant 3: Safe-call operator in combination with **let**

```
val list = listOf("1","2","3","4")
for(number in list){
        number?.let {
                println(number)
        }
}
```

Nullability

Variant 4: For all NPE Lovers, the !! operator

```
var b : String? = null
println("${b!!.length}")
```

// This will throw an NPE

```
b?.let {
        b!!.length
}
```

Variant 5: Do it yourself

```
fun <T> T?.or(default: T): T = if (this == null) default else this
```

```
var variable: String? = "Hello"
println(variable?.length.or(-1))
```

Nullability & type checks

The nullability also applies for type checks with **as** & **is**

is = instanceOf()
as = smart casting

```
var obj : Int = 1

if(obj is String){
   println(obj.length)
}
```

unsafe cast

```
val x: String = y as String
```

safe cast

```
val x: String? = y as String?
```

safe cast

```
val x: String? = y as? String
```

End of section: Nullability

Any questions ?